Model Context Protocol (MCP) Task Brief — Build Your Own LLM MCP Server

Objective:

Create a chatbot backend using **FastAPI** that acts as an **MCP-compliant server**. This backend should be capable of interfacing with **any LLM** (OpenAI, Anthropic, Mistral, etc.) via API keys provided **dynamically by the user**. Your mission is to not only make this modular and flexible but also expose your chatbot over the MCP protocol so others can **discover, connect, and communicate** with your bot.

Core Requirements:

1. **LLM Chatbot via FastAPI:**
   - Build a FastAPI chatbot that accepts prompts and returns LLM-generated responses.
   - The LLM should be user-configurable via API key at runtime. (e.g., via request headers or session config).
   - Support multiple providers (OpenAI, Anthropic, Cohere, etc.).
2. **Auth & Security:**
   - Protect your endpoints using a JWT token mechanism
   - Ensure each incoming MCP request is authenticated.
3. **MCP Server Interface:**
   - Expose your chatbot as an **MCP-compatible server**
   - Allow other MCP nodes (servers/clients) to interact with yours through this protocol.
4. **External MCP Connections:**
   - Your server should be able to connect to at least **one other MCP server** and exchange messages.
   - You're free to **imagine external MCP services** (Google Drive, Shopify, Postgres, etc) -> come up with more services that you feel are useful for a customer care center.
   - The more creative and interconnected, the better.

Bonus Points:

- Implement simple logging or tracing of the conversations.
- Include documentation or Postman collection for testing.

Submission:

- GitHub repo with:
    - Working FastAPI app
    - ReadMe with setup instructions
    - Example curl/postman scripts to test MCP interactions
    - Short video demo (Required)

Scoring Criteria (Total: 100 Points)

1. Research Depth & Tech Awareness (20 pts)

Able to dynamically handle multiple LLMs via API key.

Shows understanding of MCP or defines one if not fully clear.

Uses suitable libraries or techniques for the task (e.g., requests, pydantic, background tasks).

Evaluate and explain trade-offs between options

2. Decision-Making & External MCP Usage (20 pts)

Choose relevant external MCPs that enhance chatbot ability.

Justifies why those MCPs were chosen.

Shows creativity or utility in linking multiple MCPs.

Integration actually works (or has reasonable mocks).

3. Code Modularity & Maintainability (15 pts)

Components are broken into logical modules (e.g., router, MCP engine, LLM interface).

Code uses classes/functions well

Easily extensible: can add another LLM or MCP node with minimal change.

4. Application Structure & Design (15 pts)

Proper FastAPI project layout (routers, services, schemas, etc.).

API contracts are clean and self-explanatory (using pydantic or OpenAPI).

Secure token system in place and not hardcoded.

5. Optimization & Efficiency (10 pts)

Reuses LLM clients/sessions instead of reinitializing.

Async I/O should be properly used where needed (e.g., external API calls).

Handles exceptions & rate limits gracefully.

6. Usability, Testing & Documentation (10 pts)

Includes good README with setup + usage examples.

Bonus: Includes test cases or testable Postman collection.

App is runnable locally without jumping through flaming hoops.

7. Writing Test cases (10 pts)

At least for a few Apis or functions (unit testing and functional testing).

MCP Scenarios:

1) Google Drive MCP
   Use Case: Send Product Manual or Invoice via Chat

   When a customer requests a manual, warranty PDF, or invoice in the chat, the AI assistant (or agent) retrieves the relevant document from a pre-organized Google Drive folder and shares a secure download link.
   Use pre-shared access folders (read-only for MCP)
   Share temporary public links via Drive API